EinS:

a *Mathematica* package for computations with indexed objects

Version 2.6

Sergei A. Klioner

Institute of Applied Astronomy, Russian Academy of Sciences,
8, Zhdanovskaya St., 197042 St.Petersburg, Russia
Internet: klioner@ipa.rssi.ru

and

Lohrmann Observatorium, Technische Universität Dresden,
Mommsenstr., 13, 01062 Dresden, Germany
Internet: klioner@rcs.urz.tu-dresden.de

9 June 1997

If EinS helped you to get results which you are going to publish, a reference to an original description of the package is MANDATORY:

Klioner, S.A. (1995): EinS: a *Mathematica* package for tensorial calculations in astronomical applications of relativistic gravity theories. In Francaviglia, M. (ed.): *Abstracts of GR14, the 14th international conference on general relativity and gravitation*, p. A.182. SIGRAV-GR14, Turin

Abstract

This report contains a description of EinS version 2.6, which is a *Mathematica* package for computations with indexed objects.

Contents

1 General description

EinS is a software allowing to perform operation on indexed objects, which may or may not be tensors. EinS is a package for *Mathematica*, which is one of the most advanced computer algebra systems. EinS is a relatively small package consisting in approximately 3000 lines which constitute about 70 procedures. It was our intention to keep the package [relatively] small. The general design and functionality of EinS resulted from scientific problems in the field of astronomical applications of metric gravity theories tackled by the author during last several years. Although, the first version of EinS appeared as early as in 1994 and one can say that EinS is being developed more than 2 years, the development of EinS was not a significant separate area of our scientific activity. That is why, potential user should not expect that EinS is too general and flexible. For example, we did not try to implement built-in procedures for computing all standard quantities in General Relativity (curvature tensor, etc. (see, however, Section 4 below)). Moreover, current version of EinS does not distinguish automatically even covariant and contravariant indices. The main application field of EinS is computations with indexed objects involving implicit (Einstein) summations (EinS stands for "Einstein Summation handler"). EinS is designed to handle implicit summation rule, automatic generating of dummy indices and has an efficient algorithm for simplification of expressions containing dummy indices and objects with user-defined symmetries. The idea of the package was to create a simple, flexible package which would be easy to alter for solving any problem involving indexed objects and which would not waste time for doing anything we do not ask it to do (the latter is usual fate of the users of too general software). On the other hand, package works under *Mathematica* which is one of the most advanced and flexible computer algebra systems. This allows the user to employ all the power of *Mathematica* for solving his problem. In order to work with EinS user should have basic knowledge of *Mathematica*'s control structures and language.

EinS is not intended to perform very long computations since as well as *Mathematica* itself. The most laborious computations which we have done with the use of EinS was computing Landau-Lifshits pseudotensor for a general metric containing terms of order of $\mathcal{O}(c^{-1})$ in g_{0i}. The result contained about 2000 terms in total. Each term was a product of up to 10 indexed object, some of which were defined to symmetric. This calculation took about 6 hours on IBM PC 486DX33 with 16 Mbytes RAM. Recent progress in hardware probably allows to operate with about 5000 or more terms depending on their structure within EinS, RAM being principal limitation. However, if you have to operate with longer expressions you have to turn to STENSOR (see, e.g., [1]).

For the moment there are only two [semi-]official publications concerning EinS [4, 5]. If EinS helped you to derive a publishable results please refer to the official description of the package as described on page 2. We would also appreciate receiving a reference on such publications.

Any reports of bugs would be appreciated and I will do my best in fixing them.

2 Why one more package for indicial tensor calculations?

It is well known that there exists a dozen of software systems and packages for doing tensorial computations on a computer (see, [6, 1] for comprehensive reviews). It is necessary to distinguish between indicial tensorial computation and computation of components of tensors. These two class of packages are quite different. The latter usually allow one to compute a standard set of quantities for given components of metric: components of Christoffel symbols, components of curvature tensor etc. The examples of such packages are SHEEP (including CLASSI), EXCALC for REDUCE, CTENSR for Macsyma, and recently developed GRTensor/II for MAPLE, and TTC and CARTAN for *Mathematica*. On the other hand, the principal aim of a package for indicial tensorial computations is to handle implicit summation and to be able to simplify expressions involving dummy indices and user-defined symmetries of various objects. Among them one could mention the packages appeared in late the 1970s: ITENSR for Macsyma and STENSOR for SHEEP and two newer packages for *Mathematica*: MathTensor and RICCI. SHEEP is a special-purpose system, and cannot handle all variety of operations which may be necessary for the user. REDUCE and Macsyma are relatively old system and are becoming probably obsolete (although the latest version of Macsyma 2.2 for MS Windows looks promising its performance cannot compete with that of Maple and even with *Mathematica*). RICCI is designed for applications in differential geometry. Although it is sometimes possible to force it to do the computation similar to those described above, its performance (especially its simplification algorithm) is too slow to handle expressions containing many thousands of terms as we have in our application field. MathTensor is a very large package requiring a lot of additional hardware resources. This makes it difficult for us to use MathTensor for deriving expressions which themselves take sometimes 10 Mbytes of RAM within *Mathematica*. Besides that neither of the packages provide us with 3+1 split of indices which is so usual and convenient in our application field restricted mainly by post-Newtonian approximations of metric gravity theories. All these arguments forced us to implement our own small and flexible package to work under *Mathematica* which is one of the best modern computer algebra systems. There are not so many programs (5–7 in total)which allow one to operate with indicial tensor expressions, and every new such program is of interest. This is the reason why even a small summary published about EinS attracted certain attention and a reference on EinS appeared in a recent review on applications of computer algebra in general relativity [1].

3 Commands

Below we describe all the commands of EinS grouped by their functionality. It is a good idea to look first at the demo programs shipped with EinS in order to understand better the logic, purposes and functionality of the package. EinS have online help messages accessible during *Mathematica* session as usual. For example, the command `?DefObject` prints a help message concerning the procedure `DefObject`. The commands of EinS can be divided into the following three groups:

- **General purpose commands** which are not directly related to the purpose of the package and extends *Mathematica* functionality. We gathered these functions in Section 3.2.
- **Commands for beginners** represent a minimal set of command sufficient to perform any calculations which EinS is originally intended for (but sometimes not in the most effective way). These commands are distributed over Sections 3.3–3.8 below and tagged with the sign ♣.
- **Commands for experienced users** are commands which could be used by an experienced user who wants to improve the performance of his programs as well as to push EinS to perform some calculation for which EinS is not intended originally. These commands are distributed over the sections below and tagged with the sign

  which "warns of a dangerous bend in the train of thought". Following Donald Knuth's TEXbook we suggest not to use this commands unless you really need.
- We describe also some of the **internal commands**. A description of these commands gives user some insight to the internal representation of the data within EinS as well as allows one to write own procedures extending functionality of the package. These commands are described in Section 3.9.

3.1 Getting started

EinS is shipped in form of encoded *Mathematica* package. You can directly read it into *Mathematica* by the command

```
<<EinS.m
```

Be sure that the directory where you placed file `EinS.m` is in the search path governed by variable `$Path` within *Mathematica*. After that the EinS logo and its version appear as well as information on *Mathematica*'s version on which it is being run. If version of

Mathematica on which EinS is being run does not coincide with that on which we have tested EinS (currently 2.1, 2.2 and 3.0), a warning message is printed. You can now proceed with typing commands of either *Mathematica* itself or EinS. You cannot load EinS twice within one *Mathematica* session. If you really need for some reasons (the author would appreciate hearing about such reasons) to load EinS again, you should restart *Mathematica*'s kernel.

3.2 General purpose commands

EinS extends *Mathematica* with a few general purpose functions which are not related directly to the calculation which EinS performs. Some of them are used internally by EinS, but nevertheless can be probably useful for users.

`SeriesLength`

`SeriesLength[expression]` prints number of terms in the expression interpreting a product as one term (that is, number of monomials in the expression). It also prints the number of terms in a series separately at each power of the small parameter.

`SaveExpression`

`SaveExpression[file_name,name]` saves the unevaluated name, equality sign and the value into the specified file. `SaveExpression` appends its output to the specified file. It does not use the Mathematica's functions `Definition` or `FullDefinition`. It works faster than SaveDefinition with the same arguments, but doesn't save the whole Definition of a symbol. This command is especially suitable when you work almost out of your RAM and want to save your result. Usual *Mathematica*'s function `Save` (which saves `FullDefinition` of the specified object) invoke a search throughout all the *Mathematica*'s memory which may result in painful swapping.

See also: `SaveDefinition`

`SaveDefinition`

`SaveDefinition[file_name,name]` saves Definition of name into the specified file. `SaveDefinition` appends its output to the specified file. If you want to save the value of specific expression, it is faster to use SaveExpression with the same arguments. Note that the *Mathematica*'s function `Save` saves `FullDefinition` of the specified object.

See also: `SaveExpression`

`KSubsets`

`KSubsets[l,k]` returns all subsets of set l containing exactly k elements, ordered lexicographically.

This function has been borrowed from the package `DiscreteMath‘Combinatorica‘` written by Steven S. Skiena to make EinS independent of a larger package. The code is gratefully acknowledged.

`ExpandSeries`

`ExpandSeries[expression]` simplifies when possible nested calls to `SeriesData` which represent power series within *Mathematica*. `ExpandSeries` allows to substitute a power series for a variables which itself appear within a power series. For example,

```
ExpandSeries[Series[1/(1+c x),{c,0,2}] /.
                           x -> Series[1/(1-c),{c,0,5}]]
```

gives correct result

```
            3
1 - c + O[c].
```

3.3 How to input objects and formulas

In order to do any calculations it is necessary to input the object with which you want to work, to describe their properties and to input their definition. EinS has a number of functions allowing to do so.

♣ `$ESDimension`

`$ESDimension` is an integer positive constant defining dimension of space, in which the user wants to work. It influences simplification of the trace of the Kronecker symbol as well as the function ToComponents.

Default: `$ESDimension = 4`

♣ `DefObject`

```
DefObject[object_name,valence,"print_name",
          {i1,i2,...},{j1,j2,...},
          {l1,...},ESIndicesLow -> True]
```

defines the object of specified valence to be symmetric with respect to indices number `{i1, i2, ...}` and alternating with respect to indices number `{j1, j2, ...}`. Both list are optional and empty by default.

The procedure also defines output formats for the object. The 3rd argument defines "print name" of the object with the specified valence. This should be a *Mathematica*'s string. It is used for generating output format used by `PrintES` and any other printing utilities of *Mathematica*, as well as TeX format which is used by `EinS2TeX` and `TeXSave`. The argument is optional. By default, name of the object itself is used as its print name. User can define another print name for TeX output (for example, it is usually desirable to print objects whose name is the name of a greek character with their TeX names (`\omega` for object `omega`). It can be done manually by the user by giving command

```
EinS2TeXString[omega]="\omega"
```

However, this manually defined TeX name shadows all other print names for objects with given name irrespective of valence until it is removed manually. One can do this with the command

```
EinS2TeXString[omega]=.
```

The last two arguments of `DefObject` regulate printing of indices as superscripts or subscripts. The list {l1,l2,...} (empty by default) specifies numbers of indices which will be printed by commands `PrintES` (and other *Mathematica*'s printing utilities) and `EinS2TeX` as subscripts, while all other indices will be printed as superscripts. The 6th parameter which is also optional could be the rule `ESIndicesLow -> True`. This makes all subscripts to be superscripts and vice verse.

Note that objects with the same input name, but with different valences can have different symmetry properties, print names, and output and TeX formats. However, we do not recommend to use this opportunity unless you are an experienced *Mathematica* user, since defining several objects with the same input name and different valences could sometimes require from the user to take extra care in choosing order in which various definitions are defined. This is a drawback of EinS, but rather a consequence of the general structure of *Mathematica* language. On the contrary, defining several objects with different input names and valences, but the same print name is absolutely "safe".

`DefObject` prints detailed report of the definition of object resulted from current call.

♣ `ESIndicesLow`

`ESIndicesLow` is a parameter for the function `DefObject` controlling if indices of the object being defined will be printed as subscripts or superscripts. If `ESIndicesLow -> False` (default) all the indices which numbers are not specified in the 5th parameter of `DefObject` are printed as superscripts, while all other indices are printed as subscripts. If `ESIndicesLow -> True` indices are printed vice verse.

See also: `DefObject`, `PrintES`, `SaveTeX`

♣ `DefES`

`DefES[expression, list_of_variables_to_be_interpreted_as_dummy_indices]` returns EinS's internal representation of the expression containing Einstein (implicit) sums. The second arguments represent a list of names of variables which are to be interpreted as dummy indexes in the specified expressions. You should take care to define these names to be local names (for example, by calling `DefES` from within *Mathematica*'s `Module` where names of dummy indices are defined to be local. This is especially recommended if you make delayed definitions via `:=`. Note that after evaluation of `DefES` each dummy index received it own unique name and all the information on the name with which you named it in the input expression of DefES is lost.

See also: `ESRange`, `$ESDebug`, `CheckDummies`

♣ `ESRange`

`ESRange` is a parameter for the function `DefES` controlling the range of dummy indices. All dummy indices within EinS can be of two types: "spatial" indices taking values `1,2,...,$ESDimension-1`, and "space-time" indices taking values `0,1,2,...,$ESDimension-1`. If `ESRange -> $ESDimension` is specified as the third argument to `DefES` all the dummy indices specified in the second argument of `DefES` are "space-time" ones. If `ESRange -> $ESDimension-1` is specified (it is actually assumed by default) all the dummy indices specified in the second argument of `DefES` are "spatial" ones. If you have to specify indices of different types in a single expressions nested calls of `DefES` should be used. The information on the type of each index is used, for example, by the procedures `ToComponents` and `SplitTime` allowing to convert an implicit sum into partially or fully explicit one.

See also: `DefES`, `$ESDimension`, `ToComponents`, `SplitTime`

♣ `PD`

EinS has its own function for partial derivative. `PD[x,y]` represents partial derivative of `x` with respect to `y`. `PD[x,{a,b,c,...}]` represents multiple partial derivative of `x` with respect to `a, b, c,...` The main reason to introduce one more

function for partial derivative in complement to *Mathematica*'s `D` was to make all objects dependent on all others by default. Besides that, the user can add addition rules to treat partial derivatives within EinS by specifying rules for `PD` without danger to destroy *Mathematica*'s engine by `Unprotect`'ing `D`. Use `DeclareConstant[x]` or `DeclareIndependent[x,n,...]` to make `x` independent of all or some variables. In current version of EinS function `PD` does not simplify derivatives of built-in *Mathematica* functions, such as `Sin`.

See also: `DeclareConstant`, `DeclareIndependent`, `DefRS`

♣ `DefRS`

`DefRS[coordinate_name,time_name]` "defines" coordinates by generating simplification rules related with partial derivative `PD` as defined in EinS, and producing output as well as TEX formats.

See also: `PD`, `PrintES`, `SaveTeX`

♣ `DeclareConstant`

`DeclareConstant[object_name,valence]` declares partial derivatives `PD` of specified object (as defined in EinS) with specified nonnegative valence with respect to any variable to be zero.

See also: `PD`, `DeclareIndependent`

♣ `DeclareIndependent`

`DeclareIndependent[object_name,valence,list_of_variables]` declares partial derivatives `PD` of specified object (as defined in EinS) with specified nonnegative valence with respect to specified variables to be zero. Note that if the specified variable is the name of coordinates defined previously by `DefRS`, the rule generated by `DeclareIndependent` concerns only "spatial" coordinate. If independence of coordinate time is also desired it should be explicitly mentioned in the `list_of_variables`.

See also: `PD`, `DeclareConstant`

♣ `Delta`

`Delta[i,j]` is the Kronecker symbol. Its dimensionality can be `$ESDimension` or `$ESDimension-1`. `Delta` is printed as "δ" under *Mathematica* 3.0 notebook interface and as "`delta`" othewise. The TEX format is always "`\delta`".

See also: `$ESDimension`, `DropDelta`

♣ `LeviCivita`

`LeviCivita[i,j,...]` represents the fully antisymmetric Levi-Civita symbol. The number of arguments could be either `$ESDimension` or `$ESDimension-1`. `LeviCivita[0,1,2,...,$ESDimension]` as well as `LeviCivita[1,2,...,$ESDimension]` are equal to `1`. Besides anti-symmetry, and output ("ε" under *Mathematica* 3.0 notebook interface and "`eps`" othewise) and TEX (always "`\varepsilon`") formats, some simplification rules for 3-dimensional Levi-Civita symbols are built in EinS.

See also: `$ESDimension`

♣ `c1`

`c1` represents $1/c$, c representing normally speed of light. This object should be used as small parameter in post-Newtonian expansions. All procedures of EinS are designed to be able to work with series expansions with respect to any variable, but the object `c1` has some nice predefined output and TEX formats.

3.4 How to print expressions

Having computed an expression you may want to look at it. If you print at with *Mathematica*'s standard printing tools (like, `Print`) you will see internal EinS's names for dummy indices, which do not look "nice". EinS has a built-in procedures enabling one to print EinS's expressions in "pretty" form.

♣ `PrintES`

`PrintES[expression,{list_of_nondummy_indices}]` `PrintES` prints the expression substituting internal names for dummy indices with the strings from the list `$Indices` which are believed to be "pretty" names for them. If the "pretty" name for a dummy index coincides with actual name of a free (non-dummy) index entering in the same expression, next member of `$Indices` is used instead. Using the second argument you can effectively exclude some of the "pretty" names from `$Indices` which you prefer to use for other purposes (for example, for free indices). Second argument is optional (default is empty list).

Because of restricted printing capabilities of *Mathematica* 2.x (namely its inability to print greek characters), `PrintES` prints "space-time" dummy indices by adding simply "`'`" to each index. On the contrary, under *Mathematica* 3.0 notebook interface, space-time indices are always displayed as small greek letters. In this case the *Mathematica* strings with the symbols to be used to display space-time dummy indices (presumably greak letters) are stored in the list `$PGIndices`. The same algorithm for selecting "pretty" names is used for space-time dummy indices.

Note that two dummy indices in two different monomials may be printed with the same "pretty" name while having different internal names. Therefore, if you have printed with `PrintES` two expressions `Ex1` and `Ex2` and see visually that they are, say, equal, *Mathematica* does not necessarily automatically simplify `Ex1-Ex2` to `0`. Use `ComputeES[Ex1-Ex2]` to simplify the difference.

Two notes for Mathematica 3.0 notebook interface users:

- The new *Mathematica* 3.0 enables one to use various `FormatType`'s to display output of your programs. Although any form could be used with EinS, our experience shows that `OutputForm` is the most appropriate to work with EinS. For the *Mathematica* 3.0 notebook front end one may want to change the default settings in `Cell` menu and set `Default Output FormatType` to be `OutputForm`. In the same way you may want to change the default settings in `Cell` menu and set `Default Input FormatType` to be `InputForm` in order to see EinS input in its native form.
- Some versions of the *Mathematica* 3.0 notebook front end contain a bug which results sometimes in improper alignment of two dimensional mathematical output if the expression contains greek characters or other non-standard characters (say, superscripts and subscripts are misaligned [shifted] with respect to the symbol carrying them). This problem does not appear on print, but only on screen. The only work-around known to the author is to avoid naming object with non-standard symbols. Let us hope that Wolfram Research, Inc. will fix the bug as soon as possible.

See also: `$Indices`, `$PGIndices`, `ComputeES`

♣ `$Indices`

`$Indices` is a list of one-letter strings to be used as "pretty" names for dummy indices when EinS outputs expressions using functions `PrintES`, `SaveTeX` or `EinS2TeX`.

See also: `PrintES`, `SaveTeX`

♣ `$PGIndices`

`$PGIndices` is a list of one-letter strings to be used as "pretty" names for space-time dummy indices when EinS outputs expressions using `PrintES` under *Mathematica* 3.0 notebook interface. By default, it contains all small greak letters in alphabetical order.

See also: `PrintES`

3.5 How to compute and simplify expressions

EinS has a variety of functions aimed at various simplifications of expressions taking into considerations symmetries of user-defined as built-in objects as well as opportunity to rename dummy indices. Function `ComputeES`, which performs standard maximal possible simplification of an expression, is recommended for beginners. It always returns the simplest form of the expression, but not necessarily in minimal time. Other functions could be used by experienced users to speed up the computations.

♣ `ComputeES`

`ComputeES[expression]` performs standard maximal simplification possible within EinS. `ComputeES[expression]` is equivalent to

```
UniqueES[
   ZeroBySymmetry[
      CollectES3[
         Normalize[
            ExpandAll[
               DropDelta[
                  Normalize[
                     ExpandSeries[
                        ExpandAll[expression]]]]]]]]]
```

After standard simplification each dummy index in the resulting expression has own unique name.

If `$ESDebug===True`, `ComputeES` uses `CheckDummies` to check if each dummy index appear exactly twice in each term.

See also: `UniqueES`, `ZeroBySymmetry`, `CollectES3`, `Normalize`, `DropDelta`, `ExpandSeries`, `CheckDummies`, `$ESDebug`

`DropDelta`

`DropDelta[expression]` is a procedure simplifying terms containing Kronecker symbol `Delta[i,j]` with a dummy index (or indices).

See also: `Delta`

`CollectES`

`CollectES[expression]` tries to match Einstein sums in the expression renaming dummy indices. This function is usually slower than `CollectES2`, but seems to be useful. If `$ESDebug===True` it prints numbers of equivalent terms. `CollectES`

should be invoked only after `Normalize`. For historical reasons, `CollectES` does not account for possible antisymmetry of tensors. Procedures `CollectES`, `CollectES2` and `CollectES3` represent 3 versions of the second part of simplification algorithm implemented in EinS, the first part being `Normalize` or `MinimizeDummyIndices`, and the third one `ZeroBySimmetry`.

See also: `CollectES2`, `CollectES3`, `$ESDebug`

`CollectES2`

`CollectES2[expression]` tries to match Einstein sums in the expression renaming dummy indices. This function is much faster than `CollectES`. It uses internal Mathematica matching utility. `CollectES2` does not account for possible antisymmetry of tensors. If expression contains objects with antisymmetry as defined by `DefObject`, `CollectES2` invokes automatically `CollectES3`. Procedures `CollectES`, `CollectES2` and `CollectES3` represent 3 versions of the second part of simplification algorithm implemented in EinS, the first part being `Normalize` or `MinimizeDummyIndices`, and the third one `ZeroBySymmetry`.

See also: `CollectES3`, `CollectES`

`CollectES3`

`CollectES3[expression]` tries to match Einstein sums in the expression renaming dummy indices. `CollectES3` is equivalent to `CollectES2`, but accounts for antisymmetry of tensors. If expression does not contain objects with antisymmetry as defined by `DefObject`, `CollectES3` invokes automatically `CollectES2`. Procedures `CollectES`, `CollectES2` and `CollectES3` represent 3 versions of the second part of simplification algorithm implemented in EinS, the first part being `Normalize` or `MinimizeDummyIndices`, and the third one `ZeroBySimmetry`.

See also: `CollectES2`, `CollectES`

`UniqueES`

`UniqueES[expression]` renames all dummy indices of the expression to have unique names. It is necessary for making substitutions of `Normalize`'d expressions into other expressions.

See also: `Normalize`

`Normalize`

`Normalize[expression]` renames all dummy indices in the expression so that number of different dummy indices is minimal for each Einstein sum, and factors all terms independent of dummy indices out of the Einstein sum. Dummy indices will be

numbered in the order they appear in the expression itself (which is ordered lexicographically by *Mathematica* itself). `Normalize[expression]` is NOT equivalent to `MinimizeDummyIndices[FactorConstants[expression]]`. Procedures `Normalize` and `MinimizeDummyIndices` represent 2 versions of the first part of simplification algorithm implemented in EinS, the second part being `CollectES`, `CollectES2` or `CollectES3`, and the third one `ZeroBySimmetry`. Many terms involving implicit summation are combined automatically by *Mathematica*itself after applying `Normalize` or `MinimizeDummyIndices`.

See also: `UniqueES`, `MinimizeDummyIndices`, `FactorConstants`

`MinimizeDummyIndices`

`MinimizeDummyIndices[expression]` renames all dummy indices in the expression so that number of different dummy indices is minimal for each Einstein sum. Dummy indices will be numbered in the order they appear in the lists of dummy indices (second arguments of `ES`). Procedures `Normalize` and `MinimizeDummyIndices` represent 2 versions of the first part of simplification algorithm implemented in EinS, the second part being `CollectES`, `CollectES2` or `CollectES3`, and the third one `ZeroBySimmetry`. Many terms involving implicit summation are combined automatically by *Mathematica*itself after applying `Normalize` or `MinimizeDummyIndices`. Note that `MinimizeDummyIndices` renames dummy indices in another order a compared with `Normalize` which can be of advantage or disadvantage depending on the problem under study. `Normalize` is used in the standard simplification function `ComputeES`.

See also: `UniqueES`, `Normalize`, `ES`

`FactorConstants`

`FactorConstants[expression]` factors all terms independent of dummy indices out of each Einstein sum in the expression.

See also: `Normalize`

`ZeroBySymmetry`

`ZeroBySymmetry[expression]` tests each term of the expression for being equal to zero due to be a product of symmetric and antisymmetric factors for a pair of dummy indices. Procedure `ZeroBySymmetry` represent the third part of simplification algorithm implemented in EinS, the first part being `Normalize` or `MinimizeDummyIndices`, and the third one `CollectES`, `CollectES2` or `CollectES3`. `ZeroBySymmetry` may take relative long running time for large number of dummy indices in terms involving many objects with defined antisymmetry.

See also: `ComputeES`

3.6 How to convert implicit sums into partially and fully explicit form

EinS has two functions which allow to convert implicit summations into partially ("3+1" split in case of general relativity) or fully explicit form. The latter is a conversion from indicial tensor notations into components notations which is useful for numerical computing as well as for checking the results.

♣ `SplitTime`

`SplitTime[expression]` represents any Einstein sum involving "space-time" indices into sum of corresponding terms with zero values of indices and the sum involving "spatial" indices.

See also: `ToComponents`, `ESRange`

♣ `ToComponents`

`ToComponents[expression]` converts implicit Einstein summation into explicit form. In the explicit form all indices are integer numbers. All dummy indices are considered to run either from `0` (for "space-time" indices) or from `1` (for "spatial" indices) to `$ESDimension-1`.

See also: `SplitTime`, `$ESDimension`, `ESRange`

3.7 How to export expression into TEX

EinS has a built-in subsystem allowing to export EinS's expressions into TEX or LATEX with automatic line breaking of formulas with specified number of monomials per line.

♣ `SaveTeX`

`SaveTeX[file_name,expression,list_of_nondummy_indices]` outputs a string returned by
`EinS2TeX[expression,list_of_nondummy_indices]`
to the specified file. The meaning of the third argument is the same as for the procedure `PrintES`. The third argument is optional and empty by default.

See also: `EinS2TeX`, `$TeXDialect`

♣ `EinS2TeX`

`EinS2TeX[expression,list_of_nondummy_indices]` returns a string containing TeX form of the expression according to switches `$TeXDialect`, `$TermsPerLine`, and the list of "pretty" names for "spatial" dummy indices `$Indices` and "space-time" dummy indices `$GIndices`. The meaning of the second argument is the same

as for the procedure `PrintES`. The second argument is optional and empty by default.

See also: `SaveTeX`,`$TeXDialect`, `$TermsPerLine` `$Indices`, `$GIndices`

♣ `$GIndices`

`$GIndices` is a list of strings to be used as "pretty" names for "space-time" dummy indices when EinS outputs expressions using functions `SaveTeX` or `EinS2TeX`. By default, it contains TEX forms of small greek letters in alphabet order.

See also: `SaveTeX`, `EinS2TeX`

`EinS2TeXString`

`EinS2TeXString[expression]:=string`

A set of rules with the head `EinS2TeXString` is used when converting EinS's expressions into their TEX representation. Besides a number of predefined rules, the procedure `DefObject` automatically adds rules for user-defined objects. User can add his own rules manually which you cannot for some reason define via `DefObject`. `EinS2TeXString` should return a *Mathematica*'s string.

See also: `SaveTeX`, `EinS2TeX`, `DefObject`

♣ `$TeXDialect`

`$TeXDialect` controls whether `EinS2TeX` and `SaveTeX` output EinS expressions with alignment commands in plain TEX (`$TeXDialect===$PlainTeX`), LATEX (`$TeXDialect===$LaTeX`) or without alignment (`$TeXDialect===None`). In the latter case output is not split into lines and no alignment is performed. Default is `$LaTeX`.

♣ `$PlainTeX`

`$PlainTeX` is a possible value for `$TeXDialect`. If `$TeXDialect===$PlainTeX` `EinS2TeX` and `SaveTeX` output EinS's expressions into plain TEX form.

♣ `$LaTeX`

`$LaTeX` is possible value for `$TeXDialect`. If `$TeXDialect===$LaTeX` `EinS2TeX` and `SaveTeX` output EinS's expressions into LATEX form.

♣ `$TermsPerLine`

`$TermsPerLine` is an integer number of terms to be printed per line when formatting multiline TEX or LATEX output using functions `EinS2TeX` or `SaveTeX`.

3.8 How to debug programs

EinS has limited debugging capabilities. It includes one switch and one procedure.

♣ `$ESDebug`

`$ESDebug` is a switch turning on and off printing some additional information from several functions defined in EinS. Many of messages are of form

```
FunctionName:: ??? terms processed
```

and are printed each 100 terms. Here `FunctionName` stands for the name of the EinS's function which prints current message, and `???` for actual number of terms. These messages are useful for checking what EinS is doing now.

Beside that, if `$ESDebug===True ComputeES` and `DefES` automatically call `CheckDummies` to check the validity of the usage of dummy indices.

See also: `CheckDummies`

♣ `CheckDummies`

`CheckDummies[expression]` checks that in each term of expression each dummy index appears exactly two times and print a message otherwise. If `$ESDebug==True`, `CheckDummies` is invoked automatically by `ComputeES` and `DefES`.

See also: `$ESDebug`, `DefES`, `ComputeES`

Many of the functions generate errors or warning messages if something goes wrong. If you got an error message you have to check, for example, that you are doing a meaningful operation. If you believe that it is meaningful and cannot find an error in your code, you may want to check if each pair of dummy indices during you calculations does have a unique name. In many cases EinS tries to generate unique names automatically and it is always possible to write the program in such a way that EinS does this itself. However, if you think EinS should do this in your case, but does not do this, you may want to make this manually by using function `UniqueES`.

3.9 Some internal commands

This section describes some of the internal commands of EinS which we decided to make accessible to the users. These functions allow to write subroutines extending functionality of the package. With except for `$MaxDummy`, these functions should be used only by experienced users.

$ESVersion

A string representing EinS logo and version.

ES

ES serves as a token folding each Einstein sum.
ES[expression,list_of_dummy_indices is internal representation of Einstein sum of expression with respect to the dummy indices from the list. Normally ES should not be used by the user. Dummy indices are coded as $DummyHead[number] for ”space-time” dummy indices, and $SDummyHead[number] for ”spatial” dummy indices, where number is number of the pair of dummy indices. This number is usually unique for each pair of dummy indices processed by EinS, but can be changed by functions Normalize, UniqueES and MinimizeDummyIndices

See also: Normalize, UniqueES, MinimizeDummyIndices

STDummyQ

STDummyQ[expression] returns True if the expression is a ”space-time” dummy index and False otherwise.

See also: DummyQ, SDummyQ

SDummyQ

SDummyQ[expression] returns True if the expression is a ”spatial” dummy index and False otherwise.

See also: DummyQ, STDummyQ

DummyQ

DummyQ[expression] returns True if the expression is a dummy index and False otherwise.

See also: SDummyQ, STDummyQ

$$Alt

A symbol folding indices for which an expression is alternating.

$$Sym

A symbol folding indices for which an expression is symmetric.

♣ `$MaxDummy`

Maximal number of dummy indices to be handle by EinS. EinS performance does not depend on `$MaxDummy`. By default, `$MaxDummy=1000`.

`$DummyHead`

A string used by EinS as Head for internal representation of "space-time" dummy indices. Initially "a$$" (a *Mathematica* string).

See also: `$SDummyHead`

`$SDummyHead`

A string used by EinS as Head for internal representation of "spatial" dummy indices. Initially "i$$" (a *Mathematica* string).

See also: `$DummyHead`

4 Future prospects

As we mentioned above, EinS resulted from our own needs. It was our desire to have a relatively small, simple and easy-to-tune package allowing to perform calculations aroused from various scientific problems the author investigated. Therefore, it is generally not our intention to improve package permanently. On the contrary, the further development of the package strongly depend on the scientific problems which the author will study. Nevertheless, we list the improvements of EinS which are to be expected in the future:

- Refining simplification algorithm in two major directions:
  - automated splitting of dummy indices into subgroups which cannot intersect a priori when performing pattern matching;
  - handling more complicated symmetry properties including linear (and possibly non-linear) identities (see, e.g., [7, 2, 3]).
- Writing a special package on the basis on EinS for calculation of a standard set of quantities on the basis of metric (Christoffel symbols, curvature tensor, covariant derivative, etc.). It is clear that implementing such a package is rather trivial task having EinS. At present time, many potential pieces of such a package is distributed over various applications of EinS. However, we probably will not make EinS to distinguish automatically covariant and contravariant indices.

Besides that, some minor changes could be made on request of users, if such changes are in agreement with the author's vision of the EinS's functionality. You are encouraged to report bugs (which certainly should exist in the package) as well as suggestions to the address quoted on the title page of this document.

5 Acknowledgements

The version 2.6 of EinS described in this document has been finished while the author was Humboldt Research Fellow at the Lohrmann Observatorium, Dresden, Germany. Financial support of the Alexander von Humboldt Foundation as well as hospitality of Prof. Michael Soffel and the stuff of the Lohrmann Observatorium are gratefully acknowledged.

Index

References

[1] Hartley, D. (1996): Overview of Computer Algebra in Relativity. In: Hehl, F.W., Puntigam, R.A., Ruder, H. (eds.): *Relativity and Scientific Computing*, pp. 173–191, Springer, Berlin

[2] Ilyin, V.A., Kryukov, A.P. (1991): Symbolic Simplification of Tensor Expressions using Symmetries, Dummy Indices and Identities. In: Watt, S.M. (ed.): *ISSAC'91, Proceedings of the 1991 International Symposium on Symbolic and Algebraic Computation*, pp. 224–228, ACM Press.

[3] Ilyin, V.A., Kryukov, A.P. (1991): A symbolic simplification algorithm for tensor expressions in computer algebra. Computer Science (Programmirovanie), January 1994, pp. 83–91 (in Russian)

[4] Klioner, S.A. (1995): EinS: a *Mathematica* package for tensorial calculations in astronomical applications of relativistic gravity theories. In: *Book of Abstracts of the IAU Symposium 172: Dynamics, ephemerides and astrometry in the solar system*, p. 74, Bureau Des Longitudes, Paris

[5] Klioner, S.A. (1995): EinS: a *Mathematica* package for tensorial calculations in astronomical applications of relativistic gravity theories. In Francaviglia, M. (ed.): *Abstracts of GR14, the 14th international conference on general relativity and gravitation*, p. A.182. SIGRAV-GR14, Turin

[6] MacCallum, M.A.H. (1987): Symbolic computation in relativity theory. In: Davenport, J.H. (ed.): *EUROCAL'87, European Conference on Computer Algebra*, pp. 34–43, Springer, Berlin

[7] Rodionov, A.Ya., Taranov, A.Yu. (1987): Combinatorial aspects of simplification of algebraic expressions. In: Davenport, J.H. (ed.): *EUROCAL'87, Proceedings of the European Conference on Computer Algebra*, pp. 192–201, Springer, Berlin